Electronic structure calculation

Practices

Yongle Li

1. Hartree-Fock for atoms

1. He

Roothan-Hartree-Fock equation

$$\phi(\boldsymbol{r}) = \sum_{p=1}^{N_{tot}} C_p \chi_p(\boldsymbol{r})$$

$$\left[-\frac{1}{2}\nabla_1^2 - \frac{Z_{\mathrm{He}}}{r_1} + \sum_{r,s=1}^{N_{tot}} C_r C_s \int \mathrm{d}^3 r_2 \, \chi_r(\boldsymbol{r}_2)\chi_s(\boldsymbol{r}_2) \frac{1}{|\boldsymbol{r}_1 - \boldsymbol{r}_2|} \right] \sum_{q=1}^{N_{tot}} C_q \chi_q(\boldsymbol{r}_1) = E' \sum_{q=1}^{N_{tot}} C_q \chi_q(\boldsymbol{r}_1)$$

$$\sum_{p,q} \left(h_{pq} + \sum_{r,s} C_r C_s Q_{prqs} \right) C_q = E' \sum_{p,q} S_{pq} C_q$$

$$h_{pq} = \langle \chi_p | -\frac{1}{2}\nabla^2 - \frac{2}{r} | \chi_q \rangle \quad (Z_{\mathrm{He}} = 2)$$

$$Q_{prqs} = \int \mathrm{d}^3 r_1 \, \mathrm{d}^3 r_2 \, \chi_p(\boldsymbol{r}_1)\chi_r(\boldsymbol{r}_2) \frac{1}{|\boldsymbol{r}_1 - \boldsymbol{r}_2|} \chi_q(\boldsymbol{r}_1)\chi_s(\boldsymbol{r}_2) = [pq|rs] = \langle pr|qs \rangle$$

$$S_{pq} = \langle \chi_p | \chi_q \rangle$$

Use Gauss Type Orbital (GTO)

$$\chi_p(\boldsymbol{r}) = e^{-\alpha_p r^2}$$

$$N_{tot} = 4$$

$$\alpha_1 = 0.298073$$
$$\alpha_2 = 1.242567$$
$$\alpha_3 = 5.782948$$
$$\alpha_4 = 38.474970$$

$$S_{pq} = \int \mathrm{d}^3 r\, e^{-\alpha_p r^2} e^{-\alpha_q r^2} = \left(\frac{\pi}{\alpha_p + \alpha_q}\right)^{3/2}$$

$$T_{pq} = -\frac{1}{2}\int \mathrm{d}^3 r\, e^{-\alpha_p r^2} \nabla^2 e^{-\alpha_q r^2} = 3\frac{\alpha_p \alpha_q \pi^{3/2}}{(\alpha_p + \alpha_q)^{5/2}}$$

$$A_{pq} = -\int \mathrm{d}^3 r\, e^{-\alpha_p r^2} \frac{2}{r} e^{-\alpha_q r^2} = -\frac{4\pi}{\alpha_p + \alpha_q}$$

$$h_{pq} = T_{pq} + A_{pq}$$

$$Q_{prqs} = \frac{2\pi^{5/2}}{(\alpha_p + \alpha_q)(\alpha_r + \alpha_s)\sqrt{\alpha_p + \alpha_q + \alpha_r + \alpha_s}} = [pq\,|\,rs]$$

```
def calc_S(N_tot,Alpha):
    S = np.zeros((N_tot,N_tot))
    for p in range(N_tot):
        for q in range(N_tot):
            factor = np.pi/(Alpha[p]+Alpha[q])
            S[p,q] = factor*np.sqrt(factor)
    return S
def calc_T(Alpha,N_tot):
    T = np.zeros((N_tot,N_tot))
    for i in range(N_tot):
        for j in range(N_tot):
            y = 3.0E0*Alpha[i]*Alpha[j]*np.pi*np.sqrt(np.pi)
            z = (Alpha[i]+Alpha[j])**2*np.sqrt(Alpha[i]+Alpha[j])
            T[i,j] = y/z
    return T
def calc_A(Alpha,N_tot):
    A = np.zeros((N_tot,N_tot))
    for i in range(N_tot):
        for j in range(N_tot):
            A[i,j] = -4.0E0*np.pi/(Alpha[i]+Alpha[j])
    return A
```

S matrix

T matrix

External potential matrix

```
def calc_H(Alpha,N_tot):
    H = np.zeros((N_tot,N_tot))
    T = calc_T(Alpha,N_tot)
    A = calc_A(Alpha,N_tot)
    H = np.add(T,A)
    return H
```

Single particle Hamiltonian

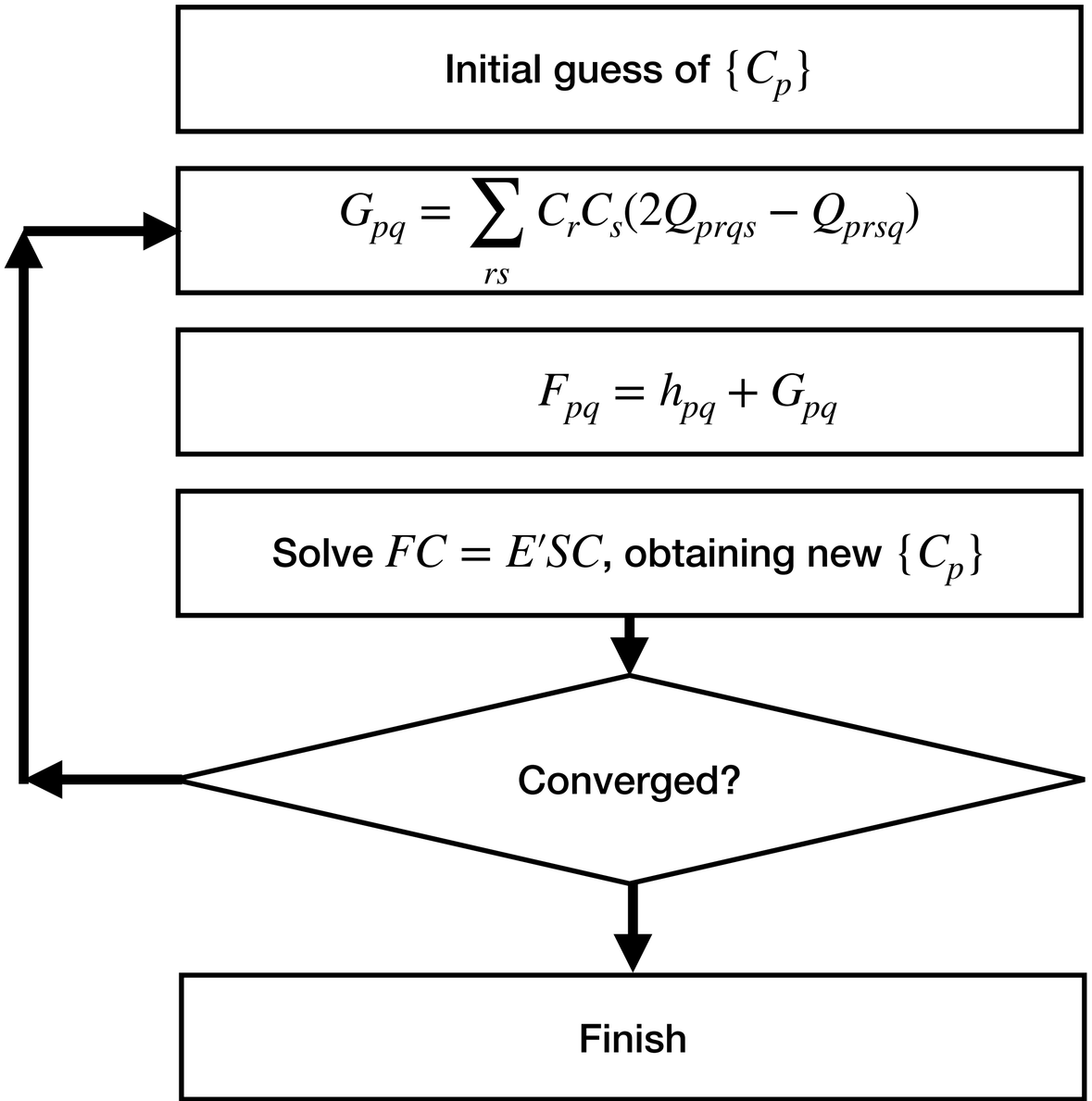
```
def calc_Q(Alpha,N_tot):
    Q = np.zeros((N_tot,N_tot,N_tot,N_tot))
    y = 2.0E0 * np.pi*np.pi * np.sqrt(np.pi)
    for r in range(N_tot):
        for s in range(r+1):
            #print("InCalcQ: r, s: ", r, s)
            for t in range(r+1):
                if t < r:
                    MaxU = t+1
                else:
                    MaxU = s+1
                for u in range(MaxU):
                    part_A = Alpha[r] + Alpha[s]
                    part_B = Alpha[t] + Alpha[u]
                    z = part_A*part_B*np.sqrt(part_A+part_B)
                    Q[r,s,t,u] = y/z
                    Q[r,s,u,t] = y/z
                    Q[t,u,r,s] = y/z
                    Q[t,u,s,r] = y/z
    return Q
```

By using symmetry, one can do only $N_{tot}^4/8$ calculations, and save some CPU time.

$$[rs\,|\,tu] = \frac{2\pi^{5/2}}{(\alpha_r+\alpha_s)(\alpha_t+\alpha_u)\sqrt{\alpha_r+\alpha_s+\alpha_t+\alpha_u}}$$

$$p \leftrightarrow q \quad r \leftrightarrow s \quad p,q \leftrightarrow r,s.$$

After obtaining the matrix elements, we can do following cycle:



The matrices are built:

T, A, H, S, Q

In practice, we introduce the density matrix

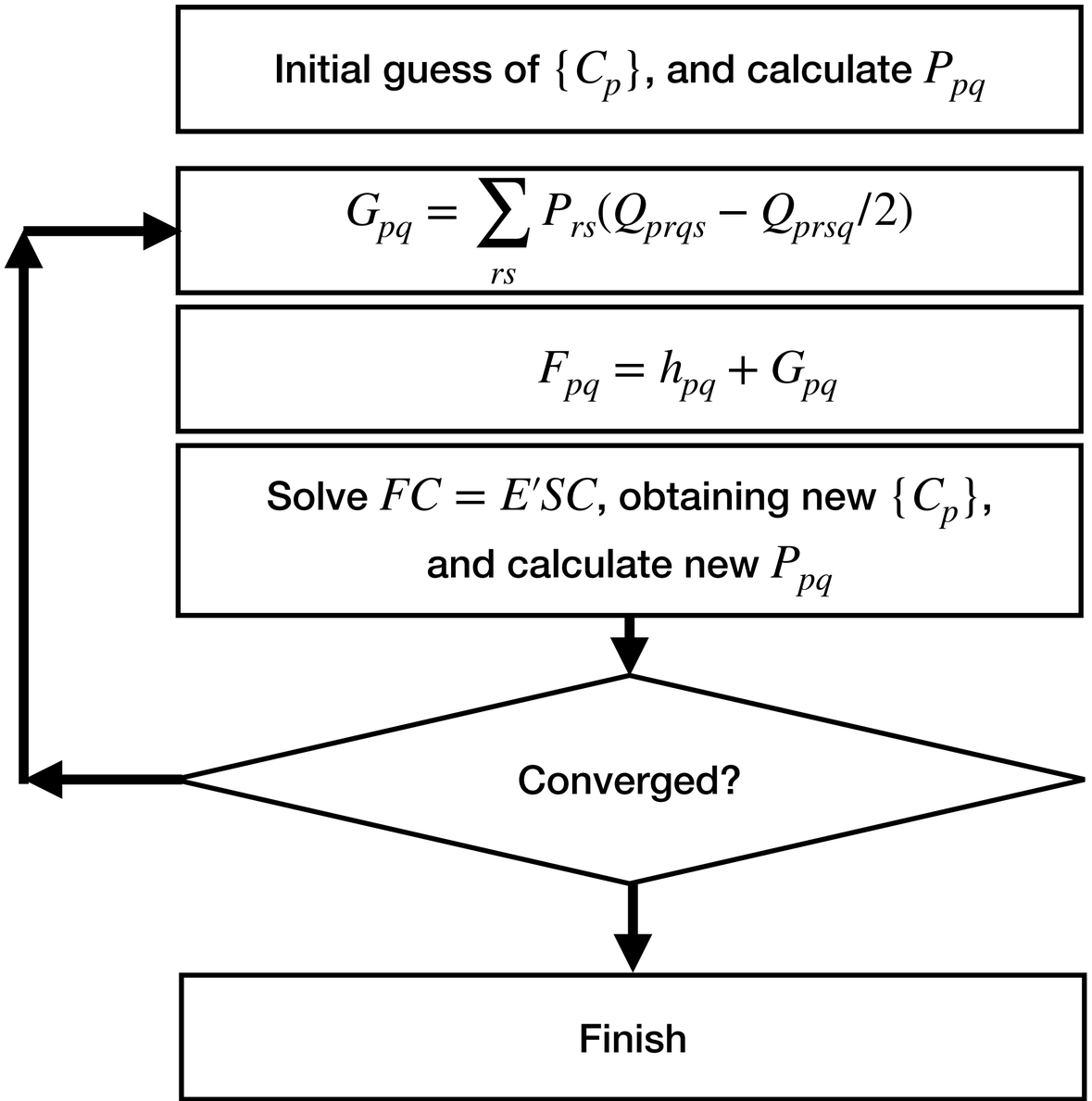
$$P_{p,q} = \frac{2CC^{\mathrm{T}}}{\mathrm{Norm}}$$

With the normalization condition:

$$\sum_{p,q=1}^{N_{tot}} C_p S_{pq} C_q = 1$$

$$\begin{aligned}\mathrm{Norm} &= \sum_{p=1}^{N}\sum_{q=1}^{N} C_p S_{pq} C_q \\ &= \sum_{p=1}^{N}\left(\sum_{q=1}^{p-1} 2C_p S_{pq} C_q + C_p S_{pp} C_p\right)\end{aligned}$$

After obtaining the matrix elements, we can do following cycle:



```
def Build_DensMat(C,S):
    '''
    D = 2 C.C^{T}/(C^{T}SC), and then normalized to:
    \sum_{i,j} 0.5*D[i,j]*S[i,j].
    Since D is symmetric, we can do it as:
    \sum_{i=1}^{N}(\sum_{j=1}^{i-1} D[i,j]*S[i,j] + 0.5*D[i,i]*S[i,i])
    '''
    DensMat = np.zeros((N_tot,N_tot))
    Norm = 0.0E0
    for r in range(N_tot):
        for s in range(r):
            DensMat[r,s] = 2.0*C[r]*C[s]
            Norm += DensMat[r,s]*S[r,s]
        DensMat[r,r] = 2.0*C[r]*C[r]
        Norm += 0.5*DensMat[r,r]*S[r,r]

    Norm = 1.0E0/Norm
    for r in range(N_tot):
        for s in range(r+1):
            DensMat[r,s] = DensMat[r,s]*Norm
            DensMat[s,r] = DensMat[r,s]
    return DensMat
```

```
def calc_G(N_tot,DensityMatrix,Q):
    '''
    Here we used the symmetry, change:
    \sum_{t=1}^{N}\sum_{u=1}^{N} (1/2)D[t,u]*Q[r,s,t,u]
    into:
    \sum_{t=1}^{N}(\sum_{u=1}^{t-1}D[t,u]*Q[r,s,t,u]+(1/2)D[t,t]*Q[r,s,t,t])
    '''
    G = np.zeros((N_tot,N_tot))
    for r in range(N_tot):
        for s in range(r+1):
            G[r,s] = 0.0E0
            for t in range(N_tot):
                for u in range(t):
                    G[r,s] = G[r,s] + DensityMatrix[t,u]*Q[r,s,t,u]
                G[r,s] += 0.5E0*DensityMatrix[t,t]*Q[r,s,t,t]
            G[s,r] = G[r,s]
    return G

def calc_F(N_tot,G,H):
    F = np.zeros((N_tot,N_tot))
    for p in range(N_tot):
        for q in range(N_tot):
            F[p,q] = H[p,q] + G[p,q]
    return F
```

According to symmetry, in the case of He atom, there are only two electrons, we have:

$$\sum_{t,u} P_{tu}([rs|tu] - \frac{1}{2}[ru|ts]) = \frac{1}{2}\sum_{t,u} P_{tu}[rs|tu]$$

Solving the generalized eigenvalue problem

$$FC = ESC$$

What we need: $V^\dagger S V = I$

$$U^\dagger S U = s$$

$$V = U s^{-1/2}$$

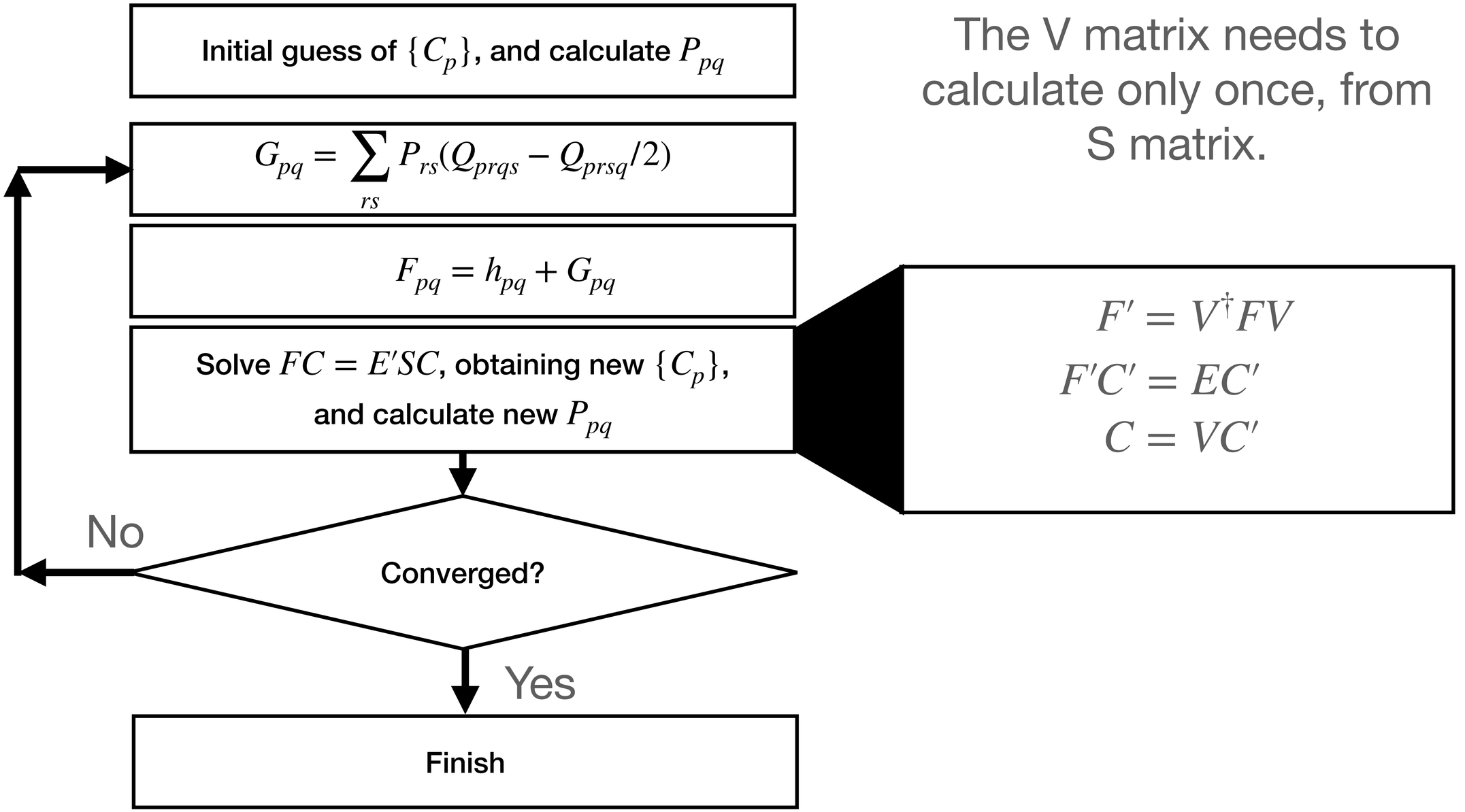
$$V^\dagger F (V V^{-1}) C = E V^\dagger S (V V^{-1}) C$$

So in practice:

$$F' = V^\dagger F V$$

$$F' C' = E C'$$

$$C = V C'$$



The V matrix needs to calculate only once, from S matrix.

```
Alpha = np.array([0.298073,1.242567,5.782948,38.47497],dtype=float)
```

```
N_tot = 4

C0=np.zeros(N_tot)
for i in range(N_tot):
    C0[i] = 1.0E0

S = calc_S(N_tot,Alpha)
D = Build_DensMat(C0,S)
```

```
H = calc_H(Alpha,N_tot)
Q = calc_Q(Alpha,N_tot)
G = calc_G(N_tot, D, Q)

small_S, U = np.linalg.eigh(S)

half_S = np.zeros((N_tot,N_tot))
for i in range(N_tot):
    half_S[i,i] = 1.0E0/np.sqrt(np.abs(small_S[i]))


V = np.dot(U,half_S)
Vd = np.transpose(V)
```

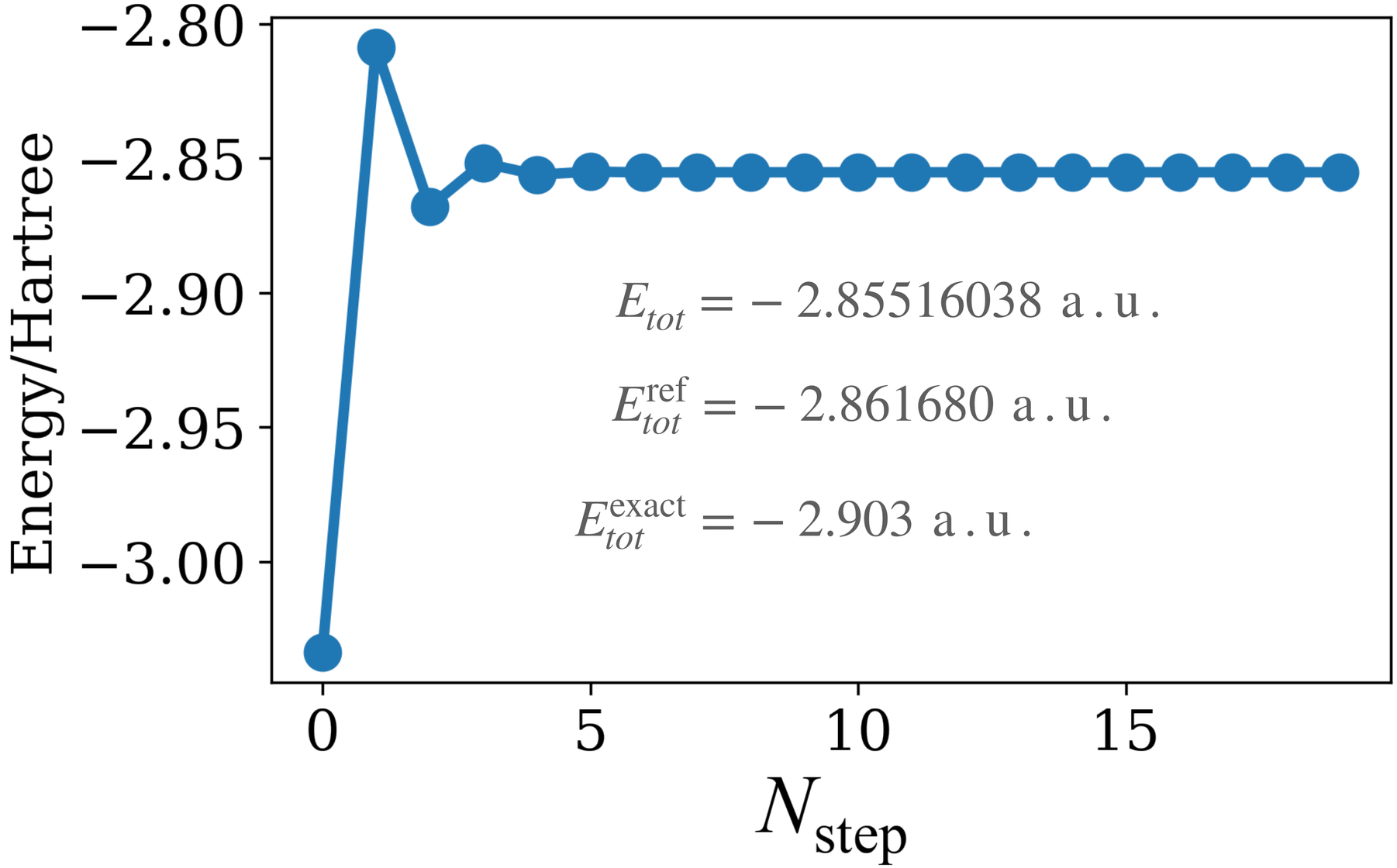
```python
def DiagFock(S,N_tot,F,H, D,V,Vd,Q,i_fact):

    tmp = np.dot(F,V) ### Here used is F.
    H_prime = np.dot(Vd,tmp)

    ### Step 5: Calculate the eigenvalue problem.
    EE, C_prime = np.linalg.eigh(H_prime)

    ## print(EE)
    C = np.dot(V,C_prime)

    C_use = copy.deepcopy(C[:,0]) ### Ground state only...
    D_new = Build_DensMat(C_use,S) ## Ground state only...

    Ener = 0.0E0
    tmp = 0.0E0

    Ener = EE[0]
    for i in range(N_tot):
        for j in range(i):
            Ener += D_new[i,j]*H[i,j]
        Ener += 0.5E0*D_new[i,i]*H[i,i]
    print("After round %4d Energy: %12.6f" % (i_fact, Ener))

    return Ener, D_new
```

$$E_{tot} = \sum_{p,q=1}^{N_{tot}} C_p C_q (H_{pq} + \frac{1}{2} F_{pq})$$

$$E_{tot} = \frac{1}{2} \sum_{p,q=1}^{N_{tot}} P_{pq} (H_{pq} + F_{pq})$$

$$E_{tot} = \sum_{p}^{N_{tot}} \left[\sum_{q=1}^{N_{tot}/2-1} P_{pq} (H_{pq} + F_{pq}) + \frac{1}{2} P_{pp} (H_{pp} + F_{pp}) \right]$$

```
OldEner = -1.0E0
Ener    = 0.0E0

ener_traj = []
for i in range(20):
    OldEner = Ener
    G = calc_G(N_tot,D,Q)
    F = calc_F(N_tot,G,H)
    Ener, D_new = DiagFock(S,N_tot,F,H, D,V,Vd,Q,i)
    ener_traj.append(Ener)
    D = copy.deepcopy(D_new)
```



1. C. C. J. Roothaan, L. M. Sachs, A. W. Weiss, *Rev. Mod. Phys*. 32:186 (1960)
2. C.-O. Almbladh and A.C. Pedroza, *Phys. Rev. A*, 29:2322 (1984)

2. Hartree-Fock for Be

```python
from pyscf import gto, scf
from functools import reduce
from scipy import linalg
```


```python
mol = gto.M(
    atom = [['Be', (0, 0, 0)]],
    basis = "sto-3g", unit="bohr")

s=mol.intor("int1e_ovlp_sph")
t=mol.intor("int1e_kin_sph")
vne=mol.intor("int1e_nuc_sph")

Hcore = np.add(t,vne)
nao=mol.nao_nr()

Q=mol.intor("int2e_sph",aosym=1)
Q=np.reshape(Q,[nao,nao,nao,nao])

N_tot = nao
```

```python
def calc_G(N_tot,DensityMatrix,Q):
    J = np.einsum("nm, nmrs -> rs", DensityMatrix,Q)
    K = np.einsum("nm, nrms -> rs", DensityMatrix,Q)
    return J - 0.5*K
```

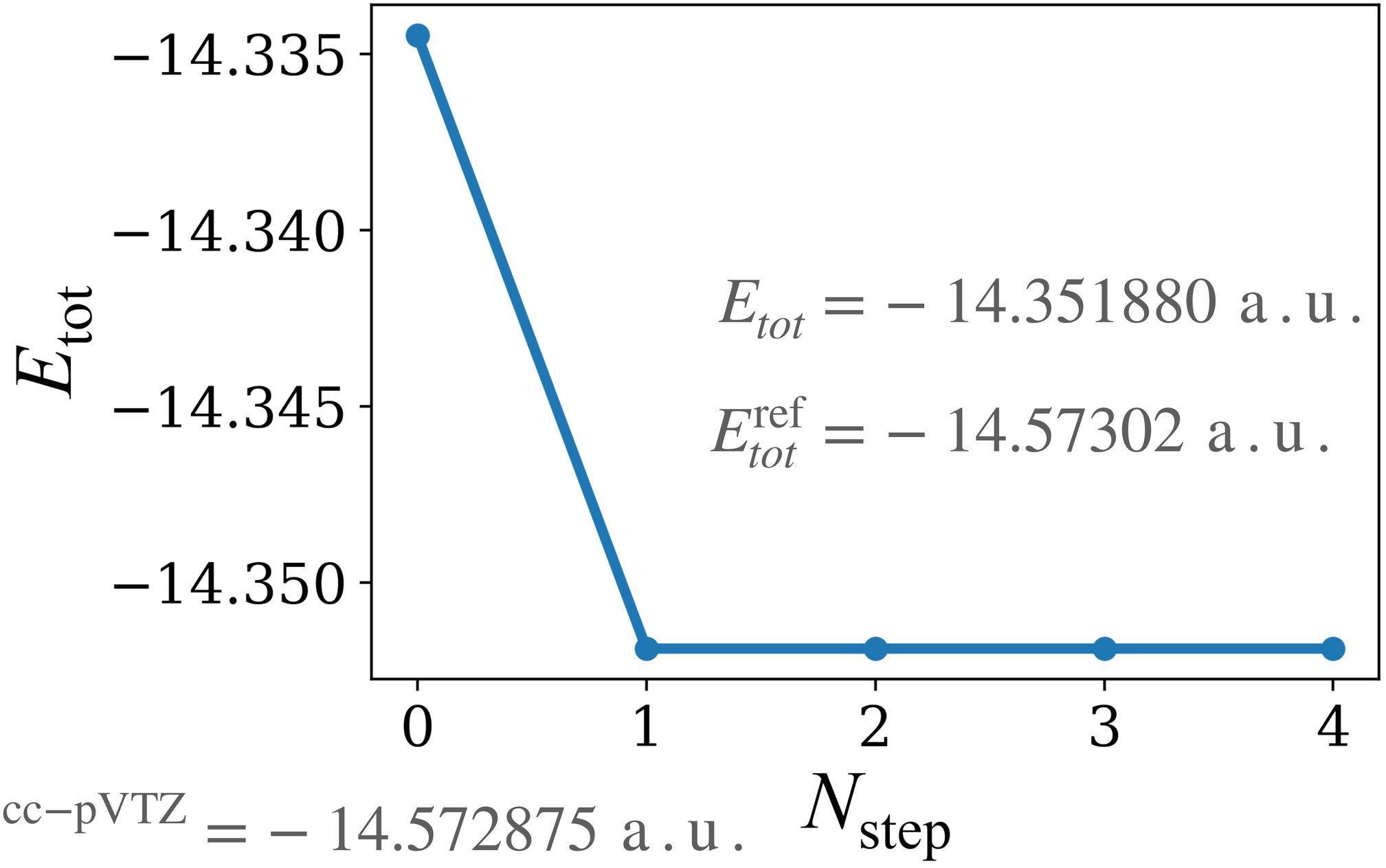
Einstein summation in NumPy

```python
def calc_F(N_tot,G,H):
    F = np.zeros((N_tot,N_tot))
    for p in range(N_tot):
        for q in range(N_tot):
            F[p,q] = H[p,q] + G[p,q]
    return F

def Build_DensMat(C,N_tot):
    mo_occ = np.zeros(N_tot)
    mo_occ[:2] = 2.0 ### Be is 1s2 2s2.
    DensMat = np.einsum("np,p,mp->nm",C,mo_occ,C)
    return DensMat
```

Here we do not normalize the density matrix.
And we use occupation number to filter out the needed part.

```
def DiagFock(S,N_tot,F,H,i_fact):

    EE, C_prime = linalg.eigh(F, S)
    idx = np.argmax(abs(C_prime.real), axis=0)
    C_prime[:,C_prime[idx,np.arange(len(EE))].real<0] *= -1

    D_new = Build_DensMat(C_prime,N_tot)

    Ener = 0.0E0
    for i in range(N_tot):
        for j in range(i):
            Ener += D_new[i,j]*(H[i,j]+F[i,j])
        Ener += 0.5E0*D_new[i,i]*(H[i,i]+F[i,i])
    print("After round %4d Energy: %12.6f" % (i_fact, Ener))

    return Ener, D_new
```

```
ener_traj = []
C = np.eye(N_tot)
D = Build_DensMat(C,N_tot)
for i in range(5):
    OldEner = Ener
    G = calc_G(N_tot,D,Q)
    F = calc_F(N_tot,G,Hcore)
    Ener, D_new = DiagFock(s,N_tot,F,Hcore, i)

    ener_traj.append(Ener)
    D = copy.deepcopy(D_new)
```



$E_{tot}^{\text{aug-cc-pVTZ}} = -14.572875$ a.u.

3. Hartree-Fock method for molecule

H_2 molecule

Matrix elements can be found in Thijssen's textbook: *Computational Physics*, or obtained from PySCF.

```
def DiagFock(S,N_tot,Fock,D,V,Vd,i_fact,Dist,H):

    tmp = np.dot(Fock,V)
    H_prime = np.dot(Vd,tmp)

    EE, C_prime = np.linalg.eigh(H_prime)

    C = np.dot(V,C_prime)
    C_use = copy.deepcopy(C[:,0])
    D_new = BuildD(S, N_tot, C_use)

    Ener = 0.0E0
    for i in range(N_tot):
        for j in range(i):
            Ener += D_new[i,j]*(H[i,j]+Fock[i,j])
        Ener += 0.5*D_new[i,i]*(H[i,i]+Fock[i,i])

    return Ener, D_new
```

$$\langle \chi_p \chi_r | g | \chi_q \chi_s \rangle = 2\sqrt{\frac{AB}{\pi(A+B)}} S_{pq} S_{sr} F_0(t).$$

Here, t is defined as

$$t = \frac{(\alpha_p + \alpha_q)(\alpha_r + \alpha_s)}{(\alpha_p + \alpha_q + \alpha_r + \alpha_s)} |\mathbf{R}_A - \mathbf{R}_B|^2;$$

$$\mathbf{R}_A = \frac{\alpha_p \mathbf{R}_p + \alpha_q \mathbf{R}_q}{\alpha_p + \alpha_q}$$

$$\mathbf{R}_B = \frac{\alpha_r \mathbf{R}_r + \alpha_s \mathbf{R}_s}{\alpha_r + \alpha_s},$$

and A and B as

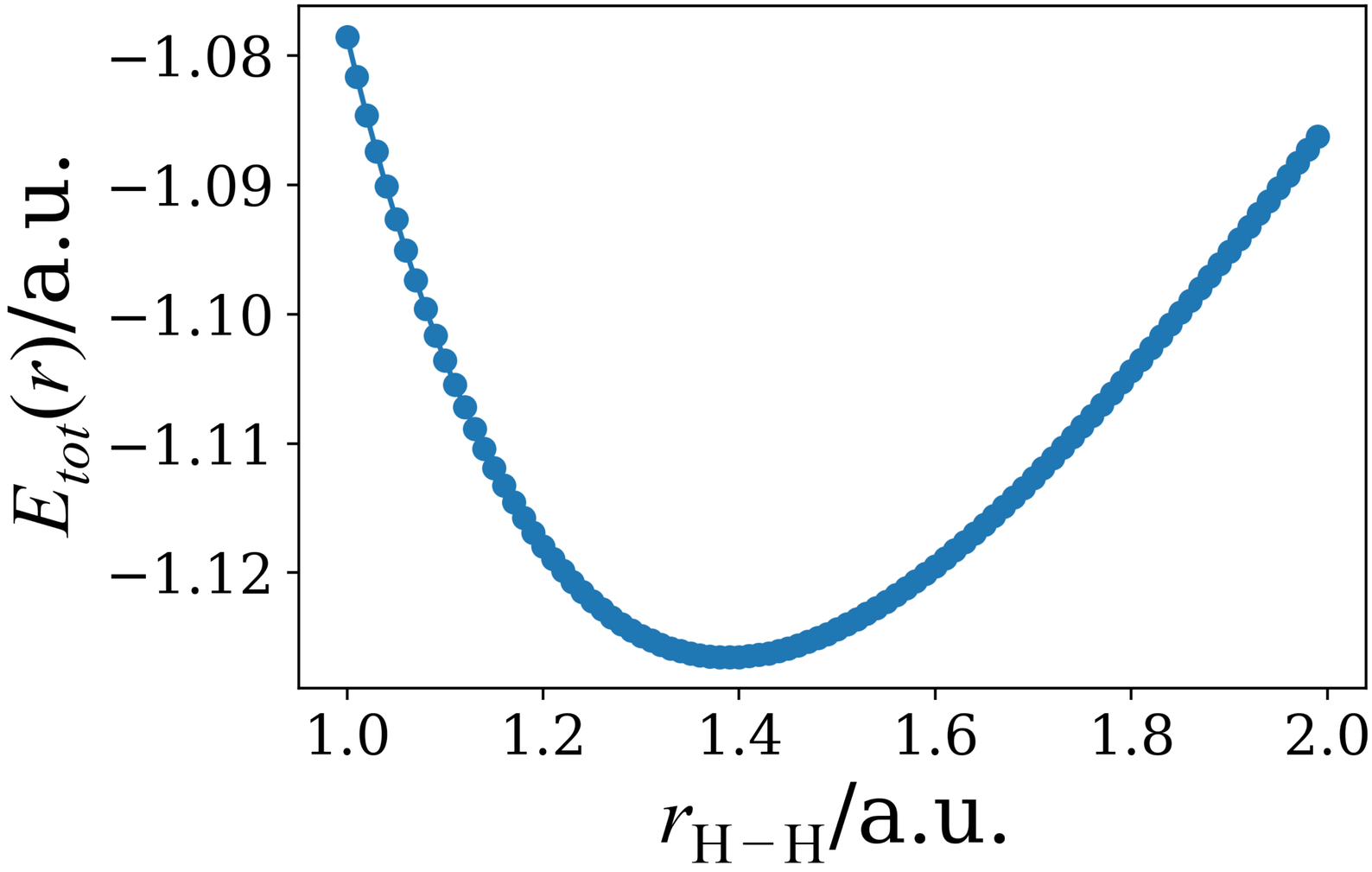
$$A = \alpha_p + \alpha_q$$

$$B = \alpha_r + \alpha_s.$$

S_{pq} is the overlap matrix.

$$F_0(x) = \int_0^1 e^{-xs^2} \mathrm{d}s$$

```
Alpha=np.array([13.00773E0,1.962079E0,0.444529E0,0.1219492E0,
        13.00773E0,1.962079E0,0.444529E0,0.1219492E0],dtype=float)
N_tot = 8
```



4. Numerov for H atom

Our problem, a homogeneous second order ODE

$$\left[-\frac{1}{2}\nabla^2 - \frac{1}{r}\right] u(r) = Eu(r)$$

To solve the ground state of H atom, since it has spherical symmetry, actually we only need to solve the radial part.

$$\nabla_r^2 = \frac{1}{r^2}\frac{\partial}{\partial r}\left[r^2\frac{\partial}{\partial r}\right] = \frac{1}{r}\frac{\partial^2}{\partial r^2}r$$

$$-\frac{1}{2r}\frac{\partial^2}{\partial r^2}[rR(r)] - \frac{1}{r}R(r) = ER(r)$$

$$-\frac{1}{2}\frac{\partial^2}{\partial r^2}[rR(r)] - \frac{1}{r}[rR(r)] = E[rR(r)] \quad (r \neq 0)$$

$$-\frac{1}{2}\frac{\partial^2}{\partial r^2}u(r) - \frac{1}{r}u(r) = Eu(r) \qquad u(r) = rR(r)$$

$$\frac{\partial^2}{\partial r^2}u(r) = f_E(r)u(r)$$

$$f_E(r) = -2(E + \frac{1}{r})$$

Here we have a homogeneous second ODE.

But the function $f(r)$ is related to the energy, which is also needed to solve.

Problem: 1) Get the eigenvalue E.

2) With E, solve $u(r) = rR(r)$.

Here to solve E one needs extra information of the ODE. Till now I know two methods:

1) Use WKB quantization.
2) Since the solving ODE with Numerov method is from a starting point to an end point, we can use the information of solution function at the end point.

WKB quantization: $\oint p\mathrm{d}q = (n + \frac{1}{2})h, \quad p = \sqrt{2m(E - V(q))}$

One needs to guess E from an initial value, calculate the integration numerically, and then see if it can fulfill some half integer with atomic unit, and then repeat this process until a match reached. (Koonin, Computational Physics, 1990)

Since the solving ODE with Numerov method is from a starting point to an end point, we can use the information of solution function at the end point.

We start from a large distance and use initial guess

$$u(r_{\max}) = r_{\max} R(r_{\max}) = r_{\max} e^{-r_{\max}}$$

And use the solution should decrease to zero at the origin point:

$$u(r = 0) = 0$$

Numerov algorithm, the major part

$$\frac{\mathrm{d}^2 u(r)}{\mathrm{d}r^2} = f(r)u(r)$$

$$u(h) + u(-h) - 2u(0) = h^2 f(0)u(0) + \frac{h^4}{12}u^{(4)}(0) + \mathcal{O}(h^{(6)})$$

From central difference:

$$u^{(4)} = \frac{u^{(2)}(h) + u^{(2)}(-h) - 2u^{(2)}(0)}{h^2}$$

Insert the differential equation $x''(t) = f(t)x(t)$

$$u^{(4)}(0) = \frac{u(h)f(h) + u(-h)f(-h) - 2u(0)f(0)}{h^2}$$

$$u(h)+u(-h)-2u(0)=h^2f(0)u(0)+\frac{h^2}{12}\left[u(h)f(h)+u(-h)f(-h)-2u(0)f(0)\right]+\mathcal{O}(h^{(6)})$$

$$\left[1-\frac{h^2}{12}f(h)\right]u(h)+\left[1-\frac{h^2}{12}f(-h)\right]u(-h)-2\left[1-\frac{h^2}{12}f(0)\right]u(0)=h^2f(0)u(0)+\mathcal{O}(h^{(6)})$$

Here we work from the $r_{\max}$ to zero , so we use the reversed working formula:

$$u(r_{i-1})=\frac{1}{\left[1-\frac{h^2}{12}f(r_{i-1})\right]}\left\{2\left[1-\frac{h^2}{12}f(r_i)\right]u(r_i)-\left[1-\frac{h^2}{12}f(r_{i+1})\right]u(r_{i+1})+h^2f(r_i)u(r_i)\right\}$$

```
def Numerov(Delta, StartI, EndI, MaxSol, FArr,\
            Sing, PhiStart, PhiNext):

    Solution = np.zeros(MaxSol)
    if (Delta<0):
        IStep = -1
    else:
        IStep = 1

    DeltaSq = Delta*Delta
    Fac = DeltaSq/12.0E0

    if (Sing):
        WPrev = PhiStart
    else:
        WPrev = (1.0-Fac*FArr[StartI-1])*PhiStart
        Solution[StartI-1] = PhiStart

    Phi = PhiNext
    Solution[StartI+IStep-1] = PhiNext
    W   = (1-Fac*FArr[StartI+IStep])*Phi

    N_start = StartI+IStep-1
    N_final = EndI - IStep
    for I in range(N_start, N_final, IStep):
        WNext = W*2.0E0 - WPrev + DeltaSq*Phi*FArr[I]
        WPrev = W
        W     = WNext
        Phi   = W/(1.0E0-Fac*FArr[I+IStep])
        Solution[I+IStep] = Phi

    return Solution
```

Here we introduce some intermediate parameters:

$$\mathrm{Fac} = h^2/12, \quad \mathrm{FArr} = f(r)$$

$$\mathrm{Phi} = u(r_i)$$

$$\mathrm{W} = \left[1 - \mathrm{Fac} * f(r_i)\right] * \mathrm{Phi}$$

$$\mathrm{WPrev} = \left[1 - \mathrm{Fac} * f(r_{i+1})\right] * u(r_{i+1})$$

$$u(r_{i-1}) = \frac{\left\{2\mathrm{W} \times \mathrm{Phi} - \mathrm{WPrev} + h^2 f(r_i)\mathrm{Phi}\right\}}{\left[1 - \frac{h^2}{12} f(r_{i-1})\right]}$$

Numerov algorithm, the boundary

Since the radial equation is singular at $r = 0$, so we use central difference to calculate the $u(r = 0)$.

Since we calculate the $u(r)$ as an array, the $u(r = 0) = u_1$

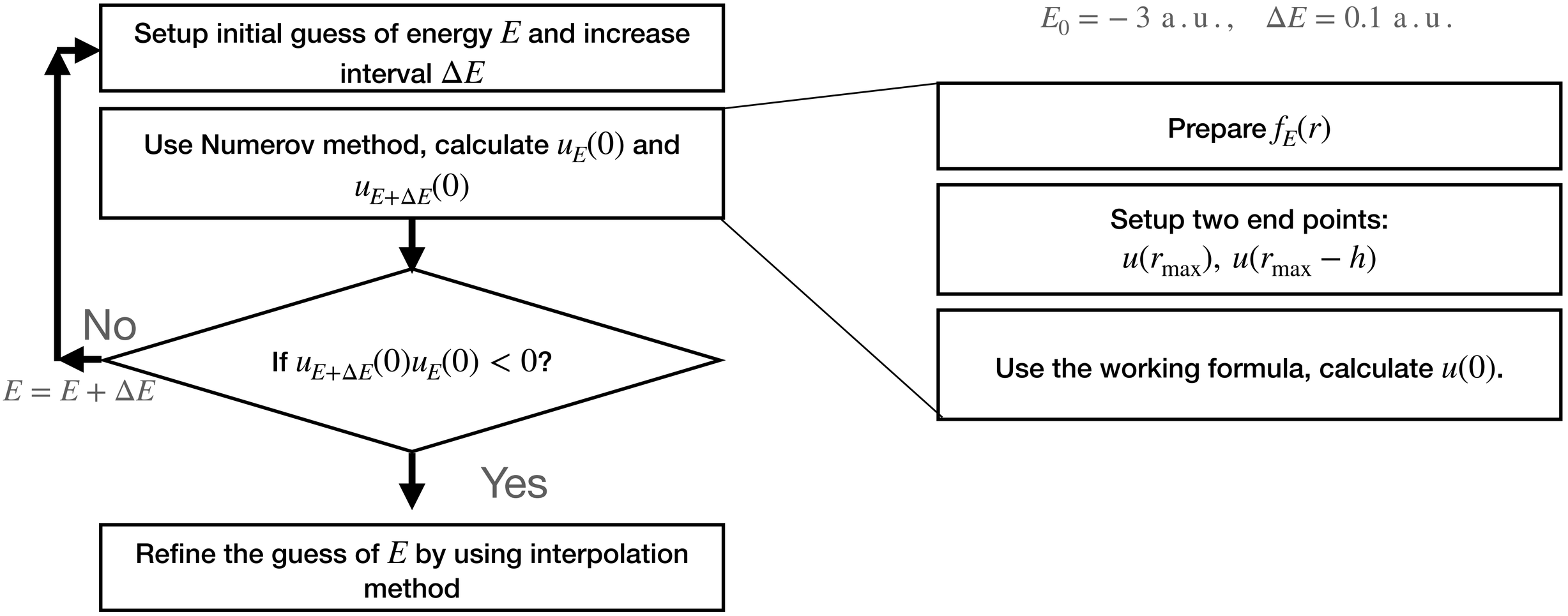
(Using Fortran format, arrays initiate from 1)

$$u_2'' = \frac{u_1 + 2u_2 - u_3}{h^2}$$

$$u_2'' \equiv u''(r_2) = f(r_2)u(r_2) = f_2 u_2$$

$$\Rightarrow u_1 = 2u_2 - u_3 + h^2 f_2 u_2$$

Working flow of Numerov method



```
def FillFArr(Energy,MaxSol,MaxI,h, Z):
    FArr = np.zeros(MaxSol)
    for I in range(1, MaxI):
        r = I*h
        FArr[I] = -2.0E0*(Energy+Z/r)
    return FArr
```

Prepare the array of $f_E(r)$

```
def PhiMax(Energy,MaxRad,MaxSol, MaxI, h, Z):

    FArr = FillFArr(Energy,MaxSol,MaxI,h, Z) ###

    PhiStart = MaxRad*np.exp(-Z*MaxRad)
    R = MaxRad-h
    PhiNext = R*np.exp(-Z*R)

    RadArr = Numerov (-h, MaxI, 0, MaxSol, FArr, False, PhiStart,\
                      PhiNext)
    Phi_Max = 2*RadArr[1]-RadArr[2]+h*h*FArr[1]*RadArr[1]
    RadArr[0] = Phi_Max

    return Phi_Max
```

Solve $u(0)$ from given $f_E(r)$.

```
def FindBound(Energy, NPrecision,MaxRad,MaxSol, MaxI, h, Z):
    Low = -3.0E0
    Step = 0.1E0
    Low = FindStep(Low, Step, PhiMax,MaxRad,MaxSol, MaxI, h, Z)
    print("Low after FindStep: %20.10f " % (Low))

    High = Low+Step
    NewPrec = NPrecision*0.10E0
    Energy = Interpolate (Low, High, NewPrec, \
                          PhiMax,MaxRad,MaxSol, MaxI, h, Z)
    print ("Energy in FindBound: %20.10f with precision of %12.2e" \
           % (Energy,NewPrec))

    return Energy
```

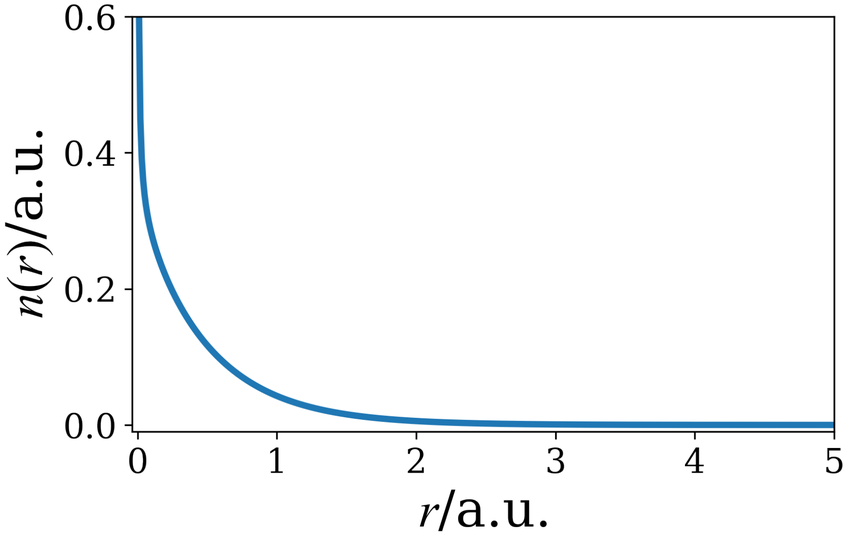
```
def FindStep (MinX, Step, PhiMax,MaxRad,MaxSol,MaxI,h,Z):

    FPrev = PhiMax(MinX,MaxRad,MaxSol, MaxI, h, Z)
    FNext = PhiMax(MinX + Step,MaxRad,MaxSol, MaxI, h, Z)
    while (FNext*FPrev > 0):
        MinX = MinX + Step
        FNext = PhiMax(MinX + Step,MaxRad,MaxSol, MaxI, h, Z)
    return MinX
```

```
def SolveRad(Energy,ChDens,MaxSol,MaxI,h,Z):
    FArr = FillFArr(Energy,MaxSol,MaxI,h, Z)
    PhiStart = MaxRad*np.exp(-Z*MaxRad)
    R = MaxRad-h
    PhiNext = R*np.exp(-Z*R)

    RadArr = Numerov (-h, MaxI, 1, MaxSol, FArr, False, PhiStart,PhiNext)
    RadArr[0] = 2*RadArr[1]-RadArr[2]+h*h*FArr[1]*RadArr[1]

    for I in range(MaxI):
        ChDens[I] = RadArr[I]**2

    NormFac = CalcInt(ChDens, MaxI, h)

    NormFac = 1.0E0/NormFac
    ChDens[0] = 0.0E0
    for I in range(1,MaxI,1):
        R = I*h
        ChDens[I] = -NormFac*ChDens[I]/R
    return Energy, ChDens
```

Once obtain a proper initial guess of E (also used Numerov method, denoted as E^{best}), we submit the E^{best} into Numerov engine again, to obtain the best guess of density.

```
MaxSol = 10000
h = 0.01
MaxRad = 10.0
Z = 1.0

MaxI = np.int(MaxRad/h)
MaxRad = MaxI*h
print(MaxRad)
Energy = -0.3E0

NPrecision = 1.0E-7
```

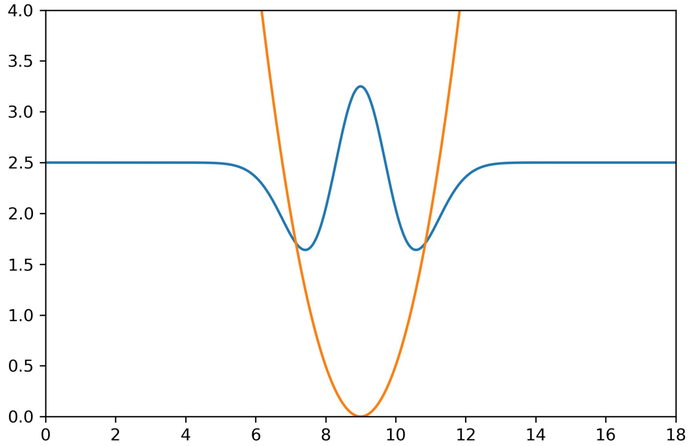
Here MaxRad is $r_{\max}$

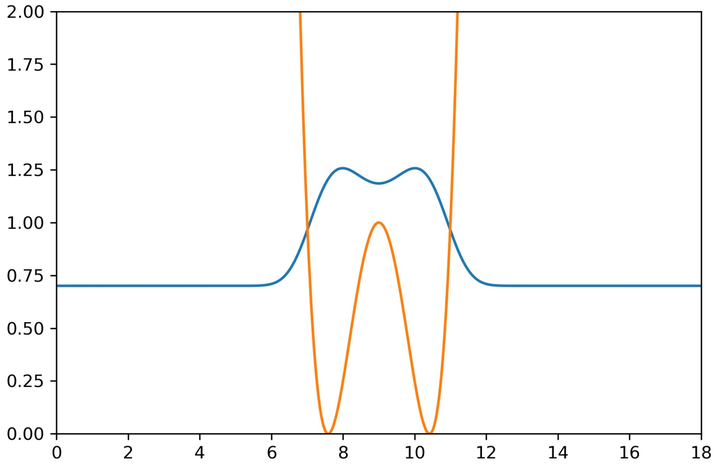


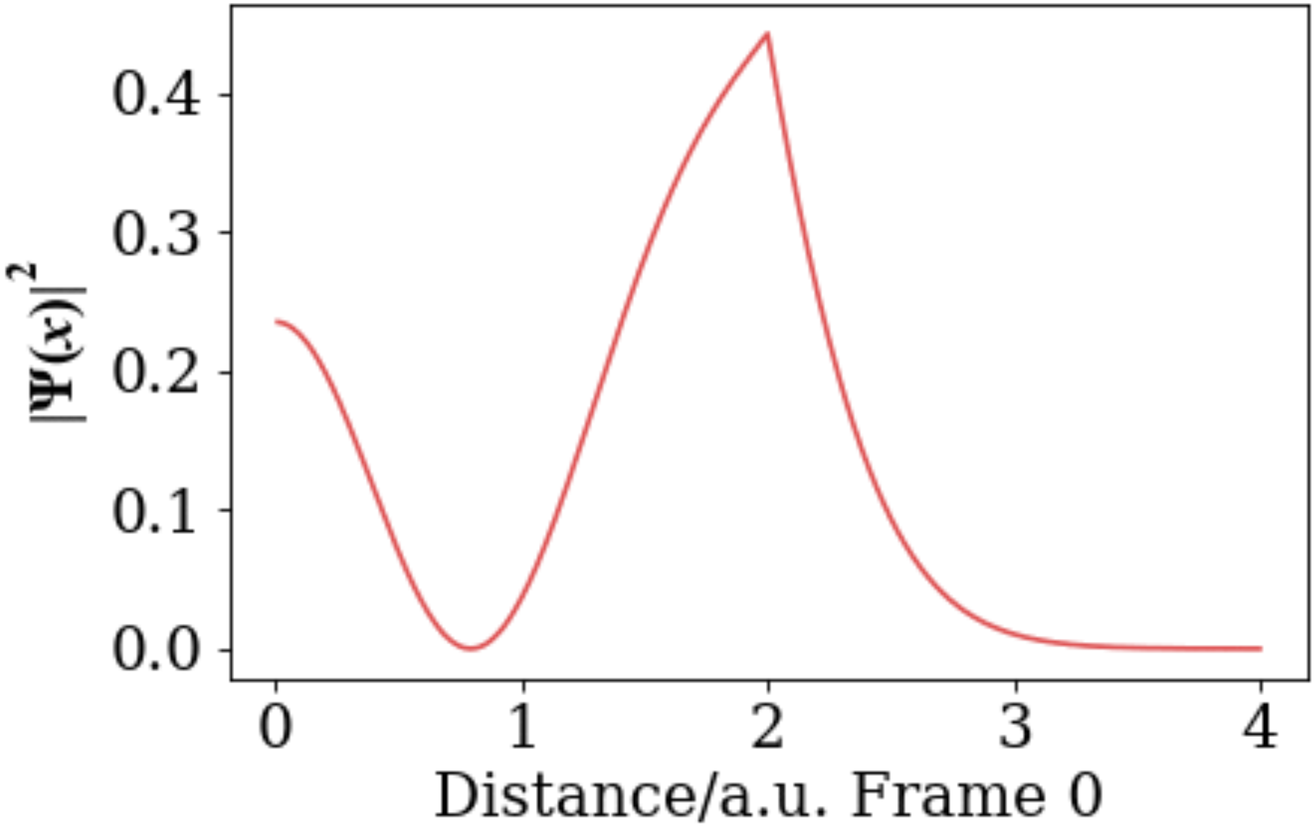

```
1  Energy = FindBound(Energy, NPrecision,MaxRad,MaxSol, MaxI, h, Z)
2  Energy, ChDens = SolveRad(Energy,ChDens,MaxSol,MaxI,h,Z)
3  print(Energy)
```

```
Low after FindStep:          -0.5000000000
Energy in FindBound:          -0.4999999802 with precision of      1.00e-08
-0.4999999802361942
```

Other applications of Numerov method







5. DFT for atom, LDA for He

He atom from DFT (LDA)

Since He has two electrons, there is a Hartree potential in the ODE.

$$\frac{\mathrm{d}^2 u(r)}{\mathrm{d}r^2} = f(r)u(r)$$

$$f(r) = -2(E + \frac{Z}{r} - V_{\mathrm{H}}(r)) = -2(E + \frac{Z}{r} - \frac{U(r)}{r}), \quad U(r) = rV_{\mathrm{H}}(r)$$

Calculation of Hartree potential

The Hartree potential is defined as: $$V_{\mathrm{H}}(\boldsymbol{r}) = \int \mathrm{d}\boldsymbol{r}' \, \frac{n(\boldsymbol{r}')}{|\boldsymbol{r} - \boldsymbol{r}'|}$$

It contains self interaction so that we do not call it Coulomb potential.
Rather than directly calculate the integration using density, we use the well-known relation in classical electrodynamics:

$$\nabla^2 \frac{1}{r} = -4\pi\delta(\boldsymbol{r})$$

$$\nabla_r^2 V_{\mathrm{H}}(\boldsymbol{r}) = \int \mathrm{d}\boldsymbol{r}'\, n(\boldsymbol{r}') \left[\nabla_r^2 \frac{1}{|\boldsymbol{r} - \boldsymbol{r}'|} \right] = -4\pi n(\boldsymbol{r})$$

Again using the same trick as $u(r) = rR(r)$, we define a new potential

$$U(r) = rV_{\mathrm{H}}(r)$$

We have the differential equation ready for being solved by Numerov method:

$$\frac{\mathrm{d}^2 U(r)}{\mathrm{d}r^2} = -4\pi r n(r)$$

This inhomogeneous ODE can also be solved using Numerov method!

Inhomogeneous Numerov method

$$\begin{cases} \dfrac{\mathrm{d}^2 U(r)}{\mathrm{d}r^2} = f(r) \\ f(r) = -4\pi r n(r) \end{cases}$$

$$x(t+h) + x(t-h) - 2x(t) = h^2 f(t) + \frac{h^2}{12}\left[f(t+h) + f(t-h) - 2f(t)\right] + \mathcal{O}(h^{(6)})$$

$$\left[x(t+h) - \frac{h^2}{12} f(t+h)\right] + \left[x(t-h) - \frac{h^2}{12} f(t-h)\right] - 2\left[x(t) - \frac{h^2}{12} f(t)\right] = h^2 f(t) + \mathcal{O}(h^{(6)})$$

$$w'(t) = x(t) - \frac{h^2}{12} f(t)$$

$$x(t-h) = 2w'(t) - w'(t+h) + h^2 f(t) + \frac{h^2}{12} f(t-h)$$

An extra trick is the Hartree potential comes from a *long range* interaction, so we need to use a switch part, ensuring it becomes integration of the product of Z/r and $n(r)$ at long range. The charge ZScr is that part of the charge of the screening electron which lies in the sphere with radius $r_{\max}$ (this charge must be close to 1 if $r_{\max}$is large).

$$Z_{\mathrm{Scr}} = 1 - (r_{\max} + 1)^2 e^{-r_{\max}}$$

$$\mathrm{HartPot}_i = U(r_i) = r_i V_{\mathrm{H}}(r_i)$$

$$\alpha = \frac{(Z_{\mathrm{Scr}} - \mathrm{HartPot}_{\mathrm{MaxI}})}{(\mathrm{MaxI} - 1)}$$

$$\mathrm{HartPot}_i = \mathrm{HartPot}_i + \alpha \times (i - 1), \quad i = 1,2,\cdots,\mathrm{MaxI}$$

Here Z_{Scr} comes from

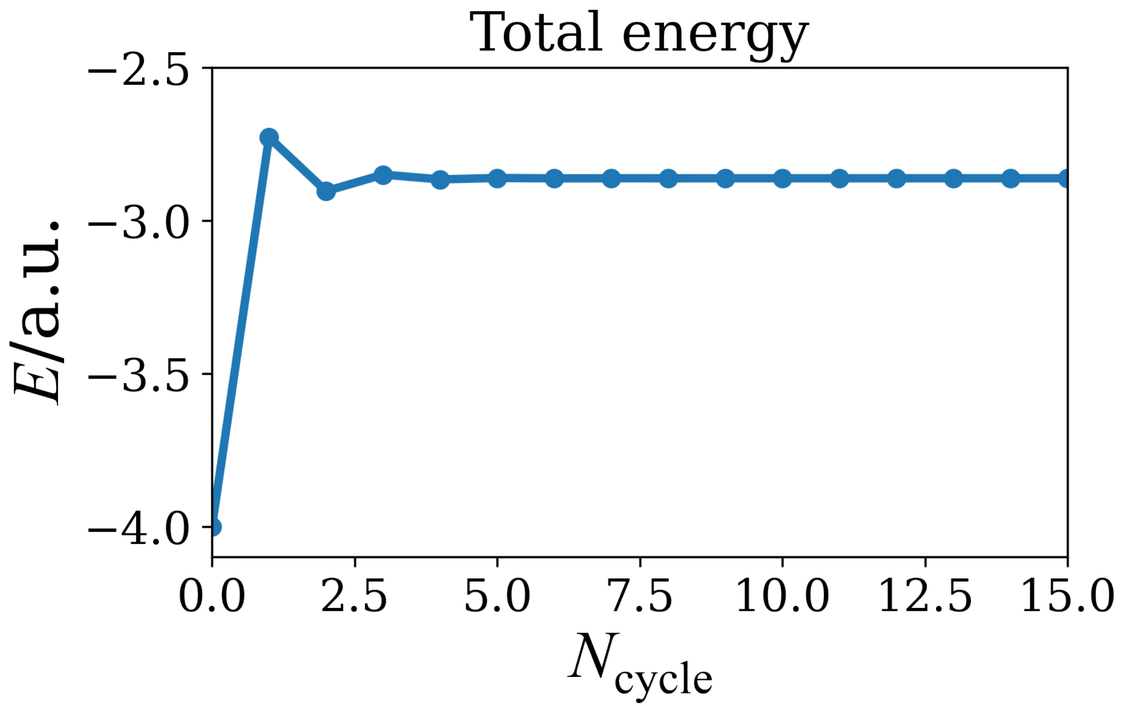
$$\lim_{r \text{ large}} \int \frac{|e^{-r'}|^2}{|\boldsymbol{r} - \boldsymbol{r}'|} \mathrm{d}^3 r'$$

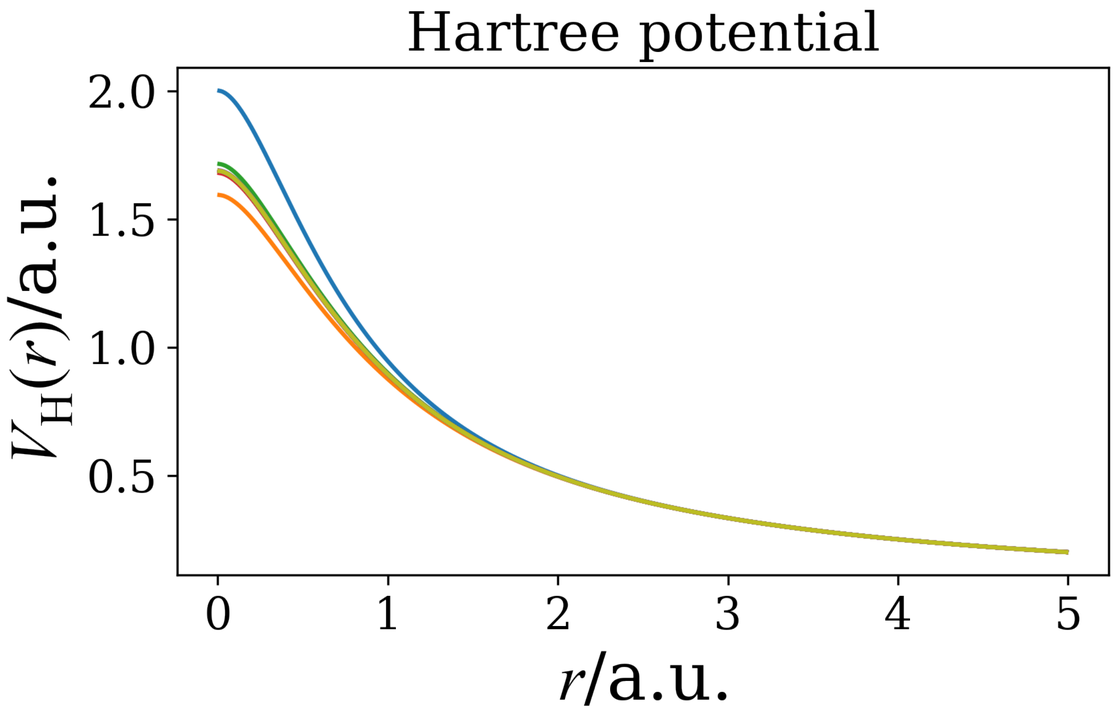
Finally, we need to make sure the wave function fulfills the normalization condition:

$$\int \mathrm{d}r\, u^2(r) = \int \mathrm{d}r\, \left[r^2 R^2(r)\right] = 1$$

```
def CalcHartPot(MaxRad,MaxSol,MaxI,h,ChDens):

    HartPot = NumInhom (h, 0, 0.0E0, MaxRad, False,0.0E0,h,ChDens, MaxSol)

    ZScr = 1.0E0-(MaxRad*MaxRad+2*MaxRad+1.0E0)*np.exp(-MaxRad)
    Alpha = (ZScr-HartPot[MaxI])/(MaxI-1)

    for I in range(MaxI):
        HartPot[I] = HartPot[I]+Alpha*I
    # print(HartPot[:4])
    return HartPot

def HartCorrect(HartPot, ChDens, MaxI, h):

    TempArr = np.zeros(MaxI)
    for i in range(MaxI):
        TempArr[i] = - HartPot[i]*ChDens[i]

    HartCorr = CalcInt(TempArr, MaxI, h)

    return HartCorr
```

```
for kk in range(20):
    OldEnergy = Energy
    Energy = FindBound(Precision,MaxRad,MaxSol, MaxI, h, Z,HartPot)
    print (Energy)
    Energy, ChDens = SolveRad(Energy,ChDens,MaxSol,MaxI,h,Z,HartPot)

    HartCorr = HartCorrect(HartPot, ChDens, MaxI, h)
    print ('Eigenvalue', Energy)
    print ('total energy',2*Energy-HartCorr)
    HartPot = CalcHartPot(MaxRad,MaxSol,MaxI,h,ChDens)
```





$$E_{tot} = -2.861513 \text{ a.u.}$$

Exchange potential of He

The exchange potential used here is Dirac's exchange from free electron gas.

$$V_{\mathrm{X}} = -\left[\frac{3}{\pi}\right]^{1/3} n^{1/3}(r)$$

$$\int \mathrm{d}r\, u^2(r) = \int \mathrm{d}r\, r^2 R^2(r) = 1$$

$$n(r) = R^2(r) = \frac{u^2(r)}{r^2}$$

$$\mathrm{ChDens} = -4\pi r n(r)$$

$$\Rightarrow$$

$$V_{\mathrm{X}}(\boldsymbol{r}) = -\left[\frac{3}{\pi}(-2)\frac{\mathrm{ChDens}}{4\pi r}\right]^{1/3}$$

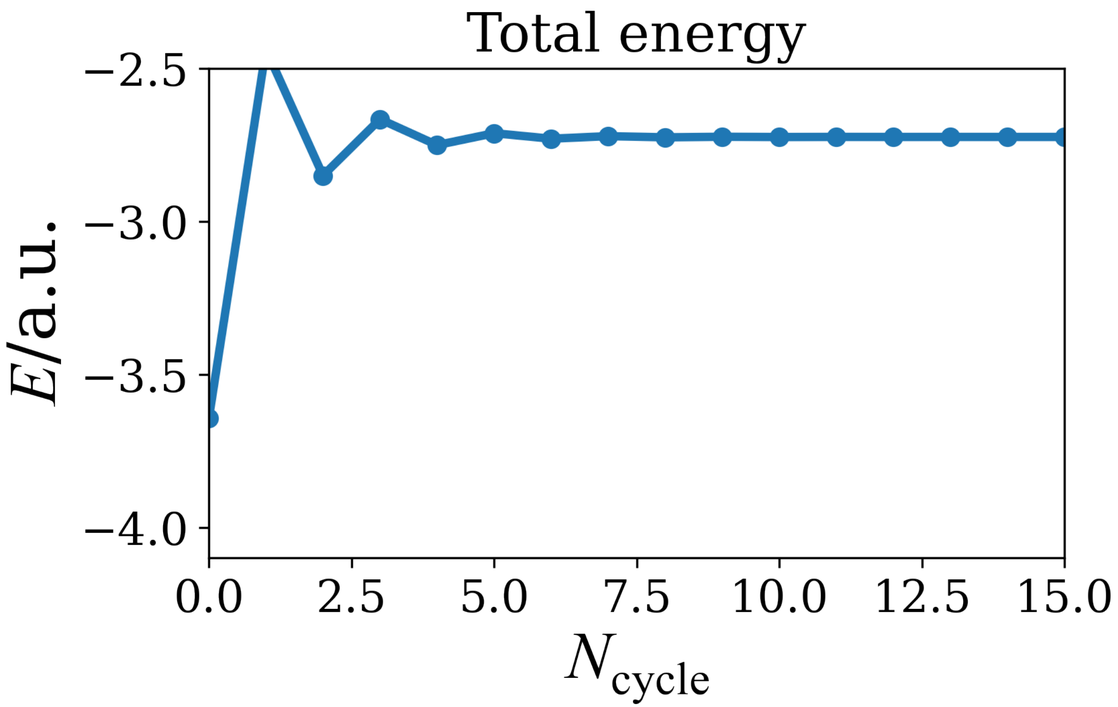
$$= -\left[-\frac{3}{2\pi^2}\frac{\mathrm{ChDens}}{r}\right]^{1/3}$$

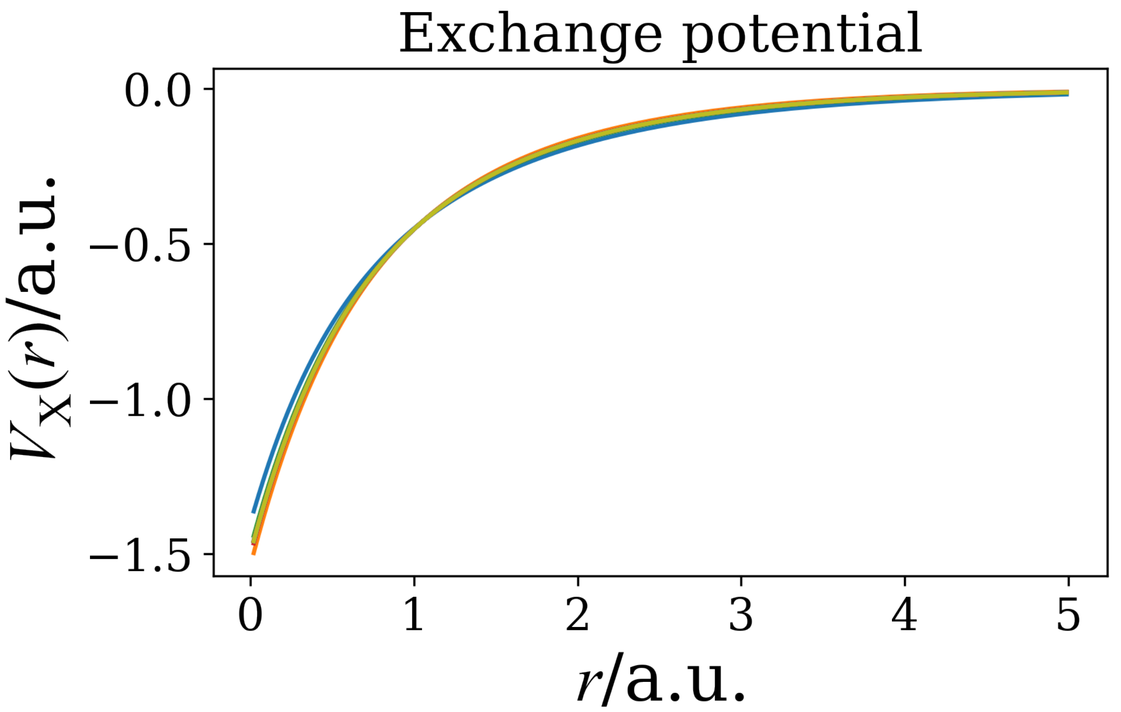
Exchange part of total energy of He

$$\varepsilon_{\mathrm{X}} - V_{\mathrm{X}} = -\left(\frac{3}{4}\right)\left[\frac{3}{\pi}\right]^{1/3} n^{1/3}(r) - \left\{-\left[\frac{3}{\pi}\right]^{1/3} n^{1/3}(r)\right\}$$

$$= -\left(\frac{1}{4}\right)\left[\frac{3}{\pi}\right]^{1/3} n^{1/3}(r)$$

$$E_{\mathrm{X}}^{\mathrm{tot}} = \int (\varepsilon_{\mathrm{X}} - V_{\mathrm{X}}) n(r) \mathrm{d}^3 r$$





$$E^{tot} = -2.723310 \text{ a.u.}$$

Correlation potential of He

The most mysterious part of DFT. In LDA, this part is fitted from high accuracy simulations of free electron gas, typically using Quantum Monte Carlo (QMC). And then the data are fitted to some complicated functions.

Here we use Perdew and Zunger's fitting formula based on data from Ceperly and Alder.

Details of PZ correlation functional

$$\varepsilon_{\mathrm{C}} = \frac{\gamma}{1 + \beta_1\sqrt{r_s} + \beta_2 r_s}, \quad r_s \geq 1$$

$$\varepsilon_{\mathrm{C}} = A \ln r_s + B + C r_s \ln r_s + D r_s, \quad r_s < 1$$

	Unpolarised	Polarised
A	0.0311	0.01555
B	−0.048	−0.0269
C	0.0020	0.0014
D	−0.0116	−0.0108
γ	−0.1423	−0.0843
β_1	1.0529	1.3981
β_2	0.3334	0.2611

$$v_{\mathrm{C}} = \varepsilon_{\mathrm{C}} \frac{1 + (7/6)\beta_1\sqrt{r_s} + (4/3)\beta_2 r_s}{1 + \beta_1\sqrt{r_s} + \beta_2 r_s}, \quad r_s \geq 1$$

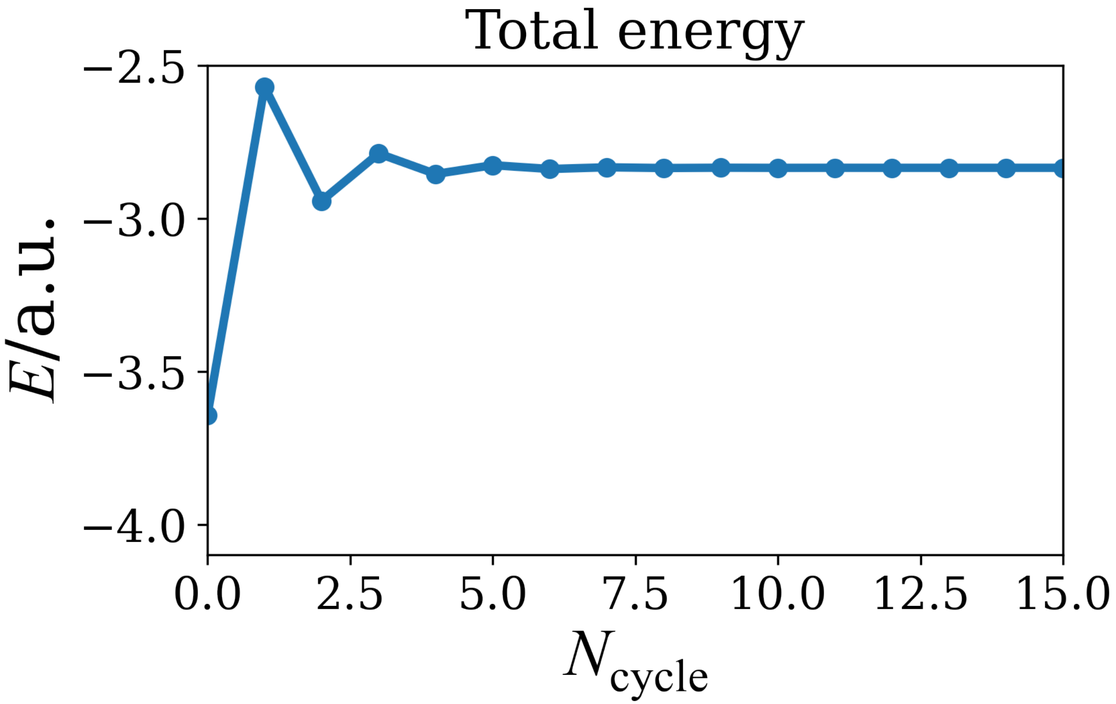
$$v_{\mathrm{C}} = A \ln r_s + B - A/3 + \frac{2}{3} C r_s \ln r_s + \frac{(2D - C)}{3} r_s, \quad r_s < 1$$

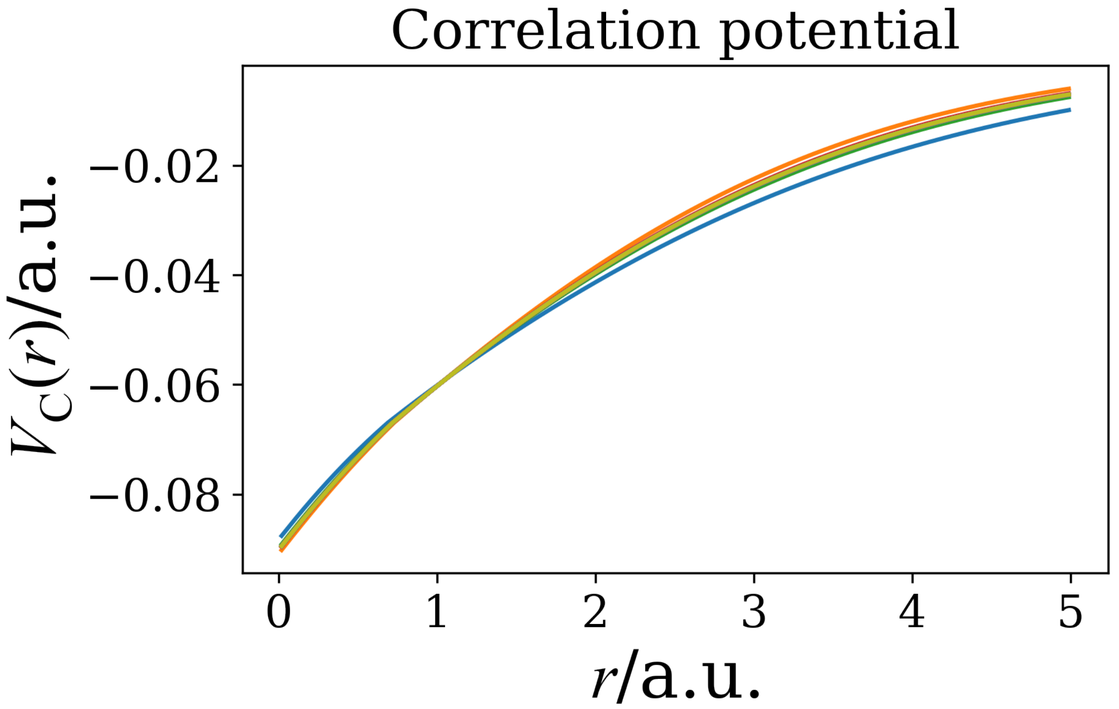
[1] J. P. Perdew and A. Zunger, *Phys. Rev. B*, 23 (1981), 5048–79.
[2] D. M. Ceperley, B. Alder, *Phys. Rev. B*, 18 (1978), 3126–38.

```python
def CeperAlder(Dens):
    '''
    Ceper-Alder correlation functional.
    '''
    Gamma = -0.1423E0
    Beta1 = 1.0529E0
    Beta2 = 0.3334E0
    A     = 0.0311E0
    B     = -0.048E0
    C     = 0.0020E0
    D     = -0.0116E0

    PI = np.pi
    Third = 1.0E0/3.0E0

    Rs = (0.75E0/(PI*Dens))**Third

    if (Rs >= 1.0E0):
        CorrEn  = Gamma/(1.0E0+Beta1*np.sqrt(Rs)+Beta2*Rs)

        CorrPot = CorrEn*(1.0E0+(7.0E0/6.0E0)*Beta1*np.sqrt(Rs)+\
                          (4.0E0/3.0E0)*Beta2*Rs)/(1+Beta1*np.sqrt(Rs)+Beta2*Rs)
    else:
        CorrEn  = A*np.log(Rs)+B+C*Rs*np.log(Rs)+D*Rs
        CorrPot = A*np.log(Rs)+(B-A/3.0E0)+(2/3.0E0)*C*Rs*np.log(Rs) + (2*D-C)*Rs/3.0E0
    return CorrEn, CorrPot
```

```
def CalcCorrPot(ChDens,MaxSol, MaxI,h):
    PI = np.pi
    Vc = np.zeros(MaxSol)
    Ec = np.zeros(MaxSol)
    Dens = -2*ChDens[1]/h
    CorrEn, CorrPot = CeperAlder(Dens)
    Vc[0] = CorrPot
    Ec[0] = CorrEn

    for I in range(1,MaxI,1):
        R = (I)*h
# Factor of 2 because of two electrons
        Dens = -2*ChDens[I]/(R*4*PI)
        CorrEn, CorrPot = CeperAlder(Dens)
        Vc[I] = CorrPot
        Ec[I] = CorrEn
    return Vc, Ec
```

```
def CorrCorrect(MaxSol,MaxI,h,ChDens,Vc,Ec):
    '''
    C Calculate \int d^r r^2 V_c(r) n (r) , which is needed in the calculation
    C of the energy.
    '''
    TempArr = np.zeros(MaxSol)

    for I in range(1,MaxI,1):
        R = (I)*h
        TempArr[I] = -(Ec[I]-Vc[I])*2*ChDens[I]*R ### This expression is from (5.21) and (5.23)
                                                  ### in J. Thijssen's book.

    CorrCorr = CalcInt(TempArr, MaxI, h)

    return CorrCorr
```





$$E_{tot} = -2.833976 \text{ a.u.}$$

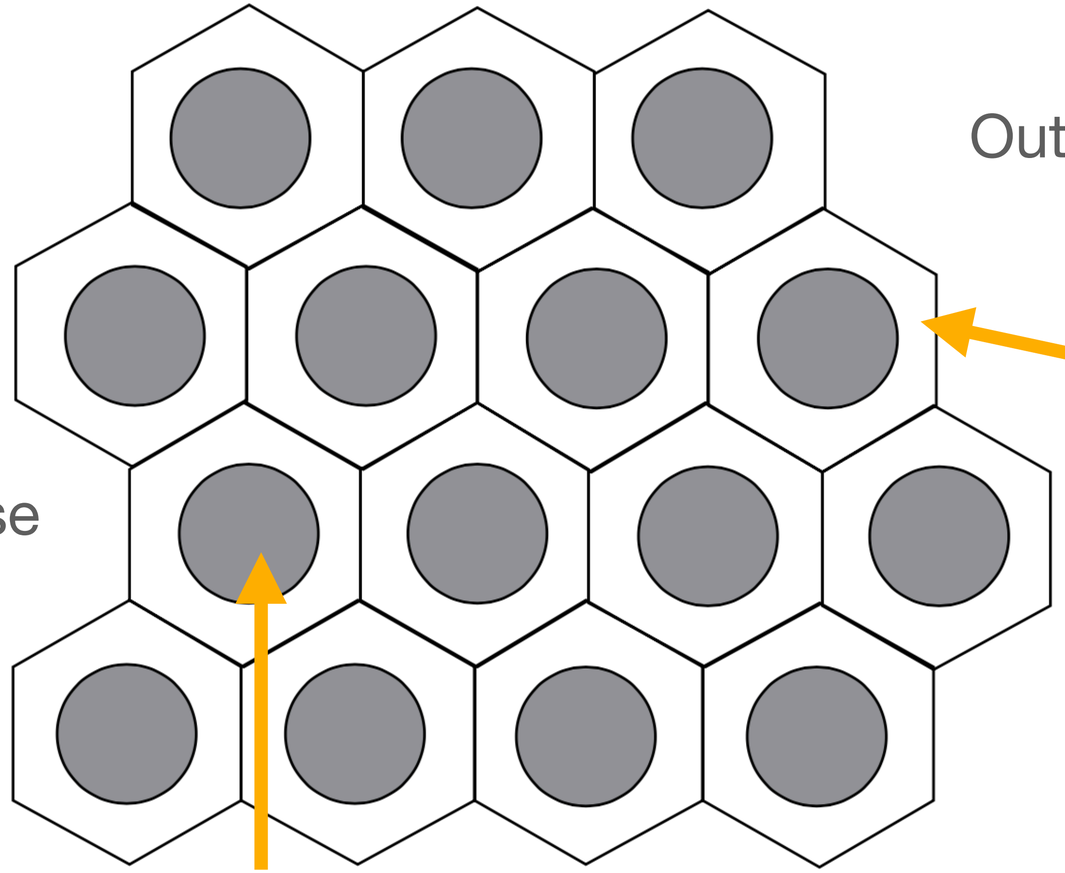
1. Set $V_{\mathrm{H}} = 0,\ V_{\mathrm{X}} = 0,\ V_{\mathrm{C}} = 0.$
2. Solve Numerov form K-S equation like in H atom, with $Z = 2$. Obtaining refined eigen energy and eigen function as an array.
3. Obtain new density from the eigen function.
4. Calculate Total energy (The eigen enegy from Kohn-Sham equation is not the total energy. Readers can refer to standard DFT books to find this.) using optimized eigen energy and Hartree potential.
5. Calculate the Hartree potential using new density.
6. Calculate the exchange functional using new density.
7. Calculate the correlation functional using new density.
8. Goto 2.

Step further

Self Interaction Correction (SIC), GGA, meta-GGA, hybrid, dispersion correction, RPA, GW, *etc*.

Augmented Plane Wave (APW)

Augmented plane wave (APW) method, also called muffin-tin potential.



Outside the muffin tin spheres, one use plane wave:

$$e^{iq\cdot r}$$

$$q = k + K$$

Inside the sphere, we use the atomic orbital:

$$\sum_{l=0}^{\infty}\sum_{m=-l}^{l} A_{lm}\mathscr{R}_l(r)Y_m^l(\theta,\phi)$$

$$\sum_{l=0}^{\infty} \sum_{m=-l}^{l} A_{lm} \mathscr{R}_l(r) Y_m^l(\theta, \phi)$$

$$\left\{ -\frac{1}{2r} \frac{\mathrm{d}^2}{\mathrm{d}r^2} r + \left[\frac{l(l+1)}{2r^2} + V(r) \right] \right\} \mathscr{R}_l(r) = E \mathscr{R}_l(r)$$

Here $V(r)$ can be exact Coulomb potential, K-S DFT functionals, or effective potential.

Again, it’s solved by Numerov method.

in APW, we match these two wave functions at the boundary, using the spherical harmonic expansion of plane wave (Messiah, *Quantum Mechanics*, Vol. 1, pp. 426)

$$e^{i\boldsymbol{q}\cdot\boldsymbol{r}} = \sum_{l=0}^{\infty} (2l+1) i^l j_l(qr) P_l(\cos(\theta'))$$

Here the angle θ' is the angle between $\boldsymbol{r}$ and $\boldsymbol{q}$.

With the help of addition formula of spherical harmonics, and the complex conjugate relation (Cohen-Tannoudji, *Quantum Mechanics*, second edition, 2020, pp. 712 (55), 714 (70)):

$$P_l(\cos(\theta')) = \frac{4\pi}{2l+1} \sum_{m=-l}^{l} (-1)^m Y^l_{-m}(\theta_{\boldsymbol{q}}, \phi_{\boldsymbol{q}}) Y^l_m(\theta, \phi)$$

$$(-1)^m Y^l_{-m} = (Y^l_m)^*$$

$$e^{i\boldsymbol{q}\cdot\boldsymbol{r}} = 4\pi \sum_{l=0}^{\infty} \sum_{m=-l}^{l} i^l j_l(qr)(Y_m^l)^*(\theta_{\boldsymbol{q}}, \phi_{\boldsymbol{q}}) Y_m^l(\theta, \phi)$$

In practice, we just use limited angular quantum numbers:

$$\sum_{l=0}^{\infty} \to \sum_{l=0}^{l_{\max}}, \quad l_{\max} = 3 \text{ or } 4$$

This matching condition fixes the coefficients A_{lm}, and we have:

$$\psi_{\boldsymbol{q}}^{\mathrm{PW}}(\boldsymbol{r}) = 4\pi \sum_{l,m} i^l \left[\frac{j_l(qR)}{\mathscr{R}_l(R)} \right] \mathscr{R}_l(r)(Y_m^l)^*(\theta_{\boldsymbol{q}}, \phi_{\boldsymbol{q}}) Y_m^l(\theta, \phi)$$

Here R is the boundary of the muffin-tin potential.

So finally with linear combination of APW wave functions, we can reach the best estimation of wave function:

$$\psi_{\boldsymbol{k}}(\boldsymbol{r}) = \sum_{\boldsymbol{K}} C_{\boldsymbol{K}} \psi^{\mathrm{PW}}_{\boldsymbol{k}+\boldsymbol{K}}(\boldsymbol{r})$$

by solving a familiar equation:

$$\mathbf{HC} = E\mathbf{SC}$$

But this is not a normal generalized eigenvalue problem, since $\mathbf{H}$ depends on energy E.

$$(\mathscr{H} - E)\mathbf{C} = 0$$

$$\mathscr{H} = \mathbf{H} - E\mathbf{S} + E\mathbf{I}$$

This problem thus changed into root-finding problem. In practice, it is hard, one can resort to Jos Thijssen's textbook. The straightforward method is to calculate its determinant, and find the position of sign-changing.

The matrix elements $\mathscr{H}_{ij}$ can be derived as:

$$\mathscr{H}_{ij} = \langle \boldsymbol{k} + \boldsymbol{K}_i | \mathscr{H} | \boldsymbol{k} + \boldsymbol{K}_j \rangle = -EA_{ij} + B_{ij} + \sum_{l=0}^{l_{\max}} C_{ijl} \frac{\mathscr{R}'_l(R)}{\mathscr{R}_l(R)}$$

$$A_{ij} = \frac{-4\pi R^2 j_l(|\boldsymbol{K}_i - \boldsymbol{K}_j| R)}{|\boldsymbol{K}_i - \boldsymbol{K}_j|} + \delta_{ij}$$

$$B_{ij} = \frac{1}{2} A_{ij} (\boldsymbol{q}_i \cdot \boldsymbol{q}_j) \quad \boldsymbol{q}_\lambda = \boldsymbol{k} + \boldsymbol{K}_\lambda,\ \lambda = i, j$$

$$C_{ijl} = (2l+1) \frac{2\pi R^2}{\Omega} P_l(\cos(\theta_{ij})) j_l(q_i R) j_l(q_j R)$$

$$\cos(\theta_{ij}) = \frac{\boldsymbol{q}_i \cdot \boldsymbol{q}_j}{q_i q_j}$$

In this practice, we try to deal with copper. It has $Z = 29$, and lattice constant $a = 6.822\ \mathrm{a.u.}$ The unit cell volume $\Omega = 3a^3/4$. Other information can be found in standard textbooks of solid state physics.

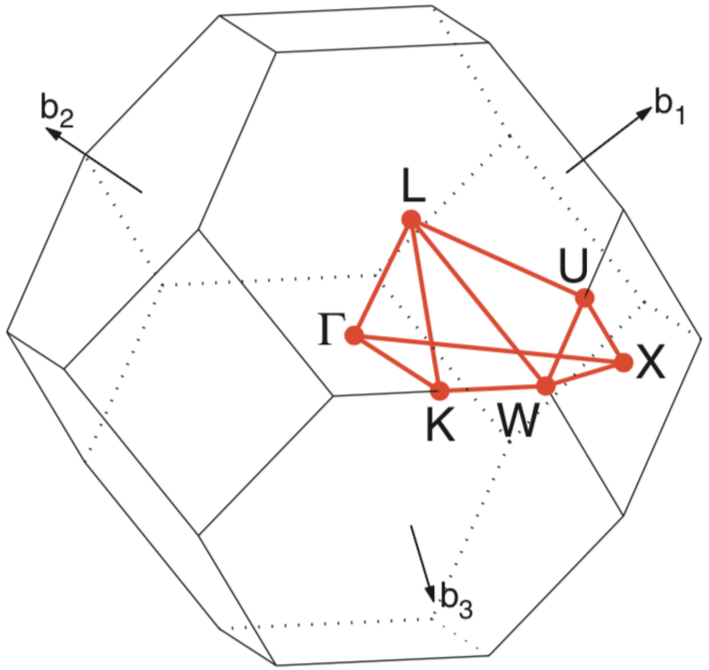


Fig. 2. Brillouin zone of FCC lattice. Path: Γ–X–W–K–Γ–L–U–W–L–$K|U$–X. An example of band structure using this path is given in Fig. 27.

W. Setyawan, S. Curtarolo, *Computational Materials Science,* **9** (2010) 299–312

$$\boldsymbol{b}_1 = \frac{2\pi}{a}(-1,1,1)^{\mathrm{T}}$$

$$\boldsymbol{b}_2 = \frac{2\pi}{a}(1,-1,1)^{\mathrm{T}}$$

$$\boldsymbol{b}_3 = \frac{2\pi}{a}(1,1,-1)^{\mathrm{T}}$$

Here we start of 27 $\boldsymbol{K} = n_i \boldsymbol{b}_i$ values, and then add its number to **113**. And beware the norm of $\boldsymbol{K}$ is:

$$|\boldsymbol{K}| = \frac{2\pi}{a}\sqrt{3n_1^2 + 3n_2^2 + 3n_3^2 - 2n_1n_2 - 2n_2n_3 - 2n_1n_3}$$

Here we also use a parametrized effective core potential:

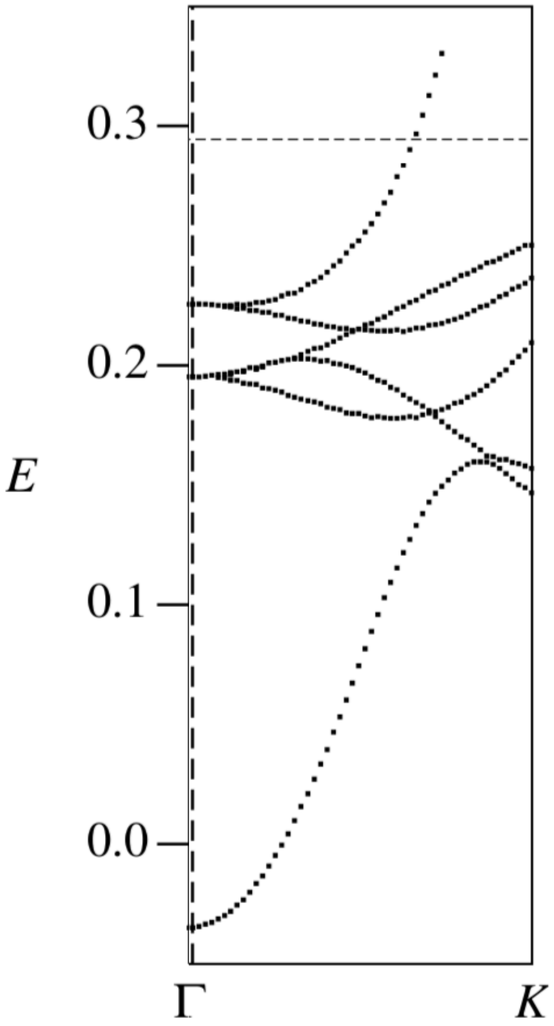
$$V(r) = Ze^{-ar^b+cr^d}(-\alpha r - \beta r^2 + \gamma r^3 - \delta r^4)$$

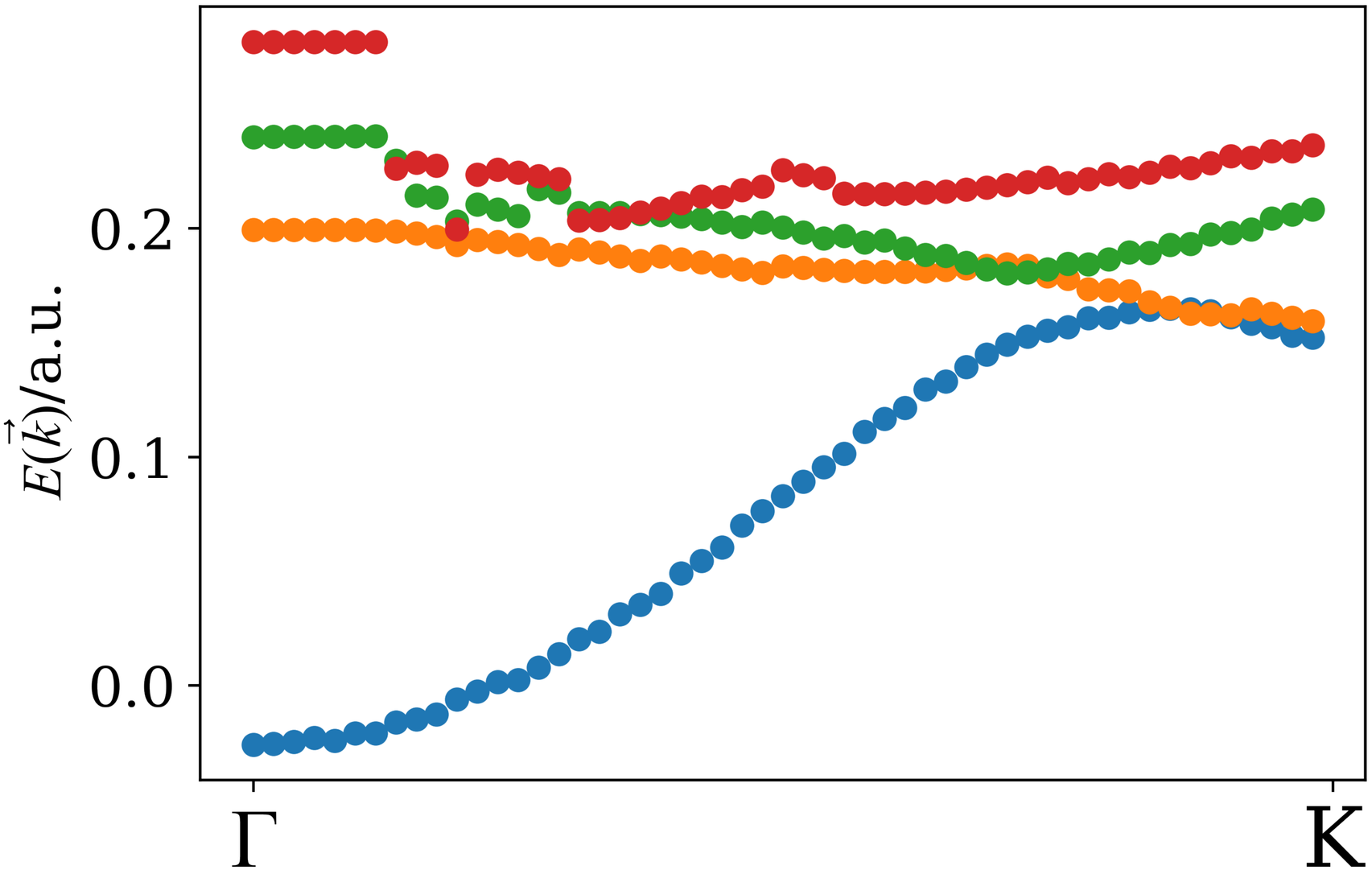
```
def Vcu(self,R):
    '''
    C Parametrisation of the potential

    Here 29 is Z_{Cu}.
    '''
    Vcu_Value = 29*np.exp(-2.3151241717834E0*R**0.8126661412243 2E0+ \
                          2.1984250222603E-02*R**4.2246376280056E0) \
    -0.15595606773483E0*R-3.1350051440417E-03*R**2+\
    5.1895222293006E-02*R**3 - 2.8027608685637E-02*R**4
    return Vcu_Value
```

Problem of the matrix elements: $\boldsymbol{q} = \boldsymbol{k} + \boldsymbol{K}$

$$\mathscr{H}_{ij} = \langle \boldsymbol{k} + \boldsymbol{K}_i | \mathscr{H} | \boldsymbol{k} + \boldsymbol{K}_j \rangle = -EA_{ij} + B_{ij} + \sum_{l=0}^{l_{\max}} C_{ijl} \frac{\mathscr{R}'_l(R)}{\mathscr{R}_l(R)}$$

Once there is only 1 atom, the $\mathscr{R}_l(R)$ tends to zero at the boundary.
One can try wave functions independent to energy (LAPW).


0.3
0.2
0.1
0.0
E
Γ
K


0.2
0.1
0.0
$E(\vec{k})$/a.u.
Γ
K

End